BBN Report-8221

Study and Simulation of Enhancements for TCP (Transmission Control Protocol) Performance Over Noisy, High-Latency Links

First Year Report

Timothy J. Shepard
Craig Partridge
Robert Coulter

BBN Technologies
GTE Internetworking
10 Moulton Street
Cambridge, MA 02138

December 1997

This effort is funded by NASA Lewis Research Center under contract number NAS3-96014 and supervised by Dr. Daniel R. Glover of NASA.

Study and Simulation of Enhancements for TCP (Transmission Control Protocol) Performance Over Noisy, High-Latency Links

—

First-Year Report

Timothy J. Shepard
Craig Partridge
Robert Coulter

—

BBN Technologies
GTE Internetworking
10 Moulton Street
Cambridge, MA 02138

December 1997

This Report documents the first year of a NASA-sponsored effort at BBN Technologies to address performance problems which are in some cases experienced by TCP (the most widely used end-to-end transport protocol in the Internet) when geosynchronous-satellite links are involved.

This effort is funded by the NASA Lewis Research Center (contract number NAS3-96014) and supervised by Dr. Daniel R. Glover (of NASA).

1 Background

The designers of the TCP/IP protocol suite explicitly included support of satellites in their design goals. The goal of the Internet Project was to design a protocol which could be layered over different networking technologies to allow them to be concatenated into an *internet*. The results of this project included two protocols, IP and TCP. IP is the protocol used by all elements in the network and it defines the standard packet format for IP datagrams. TCP is the end-to-end transport protocol commonly used between end systems on the Internet to derive a reliable bi-directional byte-pipe service from the underlying unreliable IP datagram service.

Satellite links are explicitly mentioned in Vint Cerf's 2-page article which appeared in 1980 in CCR [2] to introduce the specifications for IP and TCP. In the past fifteen years, TCP has been demonstrated to work over many differing networking technologies, including over paths including satellites links. So if satellite links were in the minds of the designers from the beginning, what is the problem? The problem is that the performance of TCP has in some cases been disappointing.

A goal of the authors of the original specification of TCP was to specify only enough behavior to ensure interoperability. The specification left a number of important decisions, in particular how much data is to be sent when, to the implementor. This was deliberately done. By leaving performance-related decisions to the implementor, this would allow the protocol TCP to be tuned and adapted to different networks and situations in the future without the need to revise the specification of the protocol, or break interoperability. Interoperability would continue while future implementations would be allowed flexibility to adapt to needs which could not be anticipated at the time of the original protocol design.

This designed-in flexibility has worked remarkably well, or is problematic, depending on your point of view. It has allowed implementors the freedom to adapt TCP to a very wide range of circumstances. For example, TCP has been tuned for situations differing by six orders of magnitude in bit rate. But this

freedom may be a mixed blessing. Potentially, there may be new bugs with every implementation. There is no complete specification of a TCP behavior that an implementor could attempt to verify a new implementation against. (The specification does not specify every state transition and every action the implementation may make.)

TCP has in reality been remarkably successful. Despite these challenges of implementation, interoperability is routinely experienced (even between two implementations which have not yet been tested against each other). That two different implementations of TCP can connect and transfer data without corruption is (today) no surprise. But occasionally, difficulties do exist in performance. How efficiently or quickly a TCP may perform may vary quite a bit as different TCPs are tried in different situations. It is in this area that research and work has continued in the development of TCP.

The most significant change in the normal practice of implementing TCP was the development in 1988 of the slow-start and congestion avoidance algorithms by Van Jacobson [3]. The key elements of these algorithms are that they start by sending a single packet, and then open a congestion window slowly as new acknowledgments arrive. When a packet is discovered to be lost, it is taken as a sign that congestion has been encountered, and the congestion window is cut in half and then grown slowly (by a single new packet per round-trip). Each new packet loss causes a new halving of the congestion window. This description of these algorithms is slightly over-simplified, but is accurate for all variants of them and captures the most important aspect of these algorithms: that they respond to congestion (as indicated by the loss of a packet) by exponentially reducing the number of packets that they allow into the net at any one time until there is no more evidence of congestion (until packets are not dropped). Then, the number of packets put into the net is increased slowly, and only as progress is made (packets are acknowledged).

These 1988 developments by Van Jacobson are today a critical part of the architecture of the Internet. These algorithms control the transmission of most of the traffic in the net today and they are in most cases the only thing keeping the net from collapsing due to congestion. They are now required of all Internet implementations by the Requirements for Internet Hosts, RFC 1122 [1].

2 Satellite links

The characteristics of satellite links include more bit-errors and increased delay (when compared with terrestrial links). That is the conventional wisdom, and is true for older satellite systems. Both of these characteristics present a problem for supporting TCP over satellite links. More bit-errors increase the number of packets lost (each of which any modern TCP will take as a sign of congestion), while the increased delay increases the number of packets that must be simultaneously in-flight to take full advantage of the available throughput that the path may carry, and reduces the rate at which TCP probes for more bandwidth (because the probing algorithm probes for one new packet per round-trip time).

Our original plan for this project was to address both of these issues. But after some thought at the beginning of the project about what level of error-correcting-code performance should be practicable to implement on wireless links with today's electronics technology, it seemed that TCP's behavior in the face of packets lost due to errors should be moot by the time we could expect to see any widespread deployment of any changes that we may be able to advocate as a result of this project. We asked "Can we expect satellite links to be practically error-free in a few years?" to satellite industry representatives at a TIA/SCD/CIS meeting in Virginia in late 1996 and later at a meeting at the NASA Lewis Research Center in early 1997, and got a near-unanimous "Yes". The two exceptions were (1) folks from one company who are today using a fairly old system (where they do see bit error rates that may cause some problems with TCP performance) and (2) from one representative concerned about military jam-resistant systems where they are concerned that under heavy jamming by the enemy, they may still see significant packet loss rates (perhaps one in ten or worse) due to uncorrectable errors on the link.

In the discussion at the TIA/SCD/CIS meeting in late fall 1996, we asked those gathered if it would be appropriate to re-direct any planned effort at making TCP cope with losses due to uncorrectable bit errors into making sure that TCP was better able to cope with long-delays (while still assuming that all losses indicate congestion) and all agreed that that should be done. So we dropped (or at least delayed indefinitely) plans to look into how to make TCP cope with error losses.

So the remaining problem that we are to address is the increased delay seen by TCPs when carried over a link involving a satellite in geosynchronous orbit. Satellites in geosynchronous orbit are a bit more than

an eighth of a light second away from ground stations on the earth. (Actually, if you are on the equator directly under the satellite, then the satellite is a bit less than an eighth of a light second above you.) Delays across terrestrial wide-area nets range from around 100 ms round-trip to cross a continent to around 400 ms round-trip between North America and Australia. The result is that the delays on paths involving single geosynchronous satellite are somewhere between 5 and 10 times longer than they would be on typical intra-continental terrestrial paths, and between 1.2 and 5 times larger than would be seen on inter-continental terrestrial paths.

3 Identifying the causes of poor performance over long-delay paths

The product of the bit rate (at which one wishes to communicate) and the round-trip delay measures the number of bits that must be "in-flight" (sent but not yet acknowledged at the sender) at once. (And dividing this by the number of bits carried per packet gives the number of packets that must be in-flight.) The TCP transmitter controls when packets are sent. It is constrained by the end-to-end flow-control window (which is carried alongside the acknowledgement in the TCP header of the returning packets). It is also constrained by the slow-start and congestion avoidance algorithms (which are included in most all TCP implementations today). What is controlled by them is the number of packets which may be in flight at any given time. If the delay is increased while the bit-rate is held constant, the number of in-flight packets must increase to maintain the same level of throughput.

Our understanding as this project began included recalling numerous anecdotal reports that TCP does not work well over long-delay paths and that the reason it doesn't work well is that the window is too small. But it is difficult to be sure of the real cause of these problems from these anecdotal reports. (Clearly, a window that is too small will limit throughput, but it is not clear that it is the only cause of trouble). There are quite a few implementations of TCP, and there are many possible variations in implementation which could impact performance (in general) and performance over long-delay paths. But we believed the most important issues were the end-to-end flow-control window and the behavior of the slow-start and congestion avoidance algorithms.

4 Simulation

To demonstrate and experiment with the effects of these algorithms, we sought a simulation environment, preferably one which already implements TCP, preferably in ways that closely follow the various versions of the BSD TCP. The BSD TCP implementations are significant because they are de-facto reference implementations. Two candidate simulation environments were considered. The first was the Netsim system developed by David Clark's group at MIT. Its advantages included the good use of a graphical display (which makes it possible to watch things as the simulation runs, and which makes it nice for running demos) and that actual BSD kernel TCP implementations had been ported to run in the Netsim simulator. Actually running real TCP implementations in the simulator is a good way to make sure that the simulation has not missed an important effect due to over simplification, and we saw this as very desirable.

The other simulator we considered, and subsequently chose, was the NS simulator developed by the network research group at LBL. Its advantages included much greater popularity in the research community, easier configuration and setup of simulation runs (using the TCL scripting language) and a larger suite of TCP simulation modules modeling many of variants of TCP that have been developed for BSD by the Internet research community in the past few years. The disadvantages of the NS simulator (in our opinion) were that the TCP modules were not constructed from actual TCP code from real implementations, and the simulation of TCP is only approximate. In the NS simulator, the simulated packets are not even real TCP packets, but are just messages carrying the information necessary to simulate the aspects of the TCP that the simulation designer thought important. Real TCP packets carry sequence numbers which represent numbered bytes, while all the TCP implementations written for the NS simulator (so far) use packet sequence numbers. The TCP implementations in the NS simulator are one-way connections, while real TCPs are full-duplex. Also, the TCP implementations in the NS simulator have vastly simpler control structures than real TCP implementations, as real TCP implementations have to handle the setup and teardown phases of TCP, and communicate the end-to-end flow control window. The TCPs in NS do not even carry the end-to-end

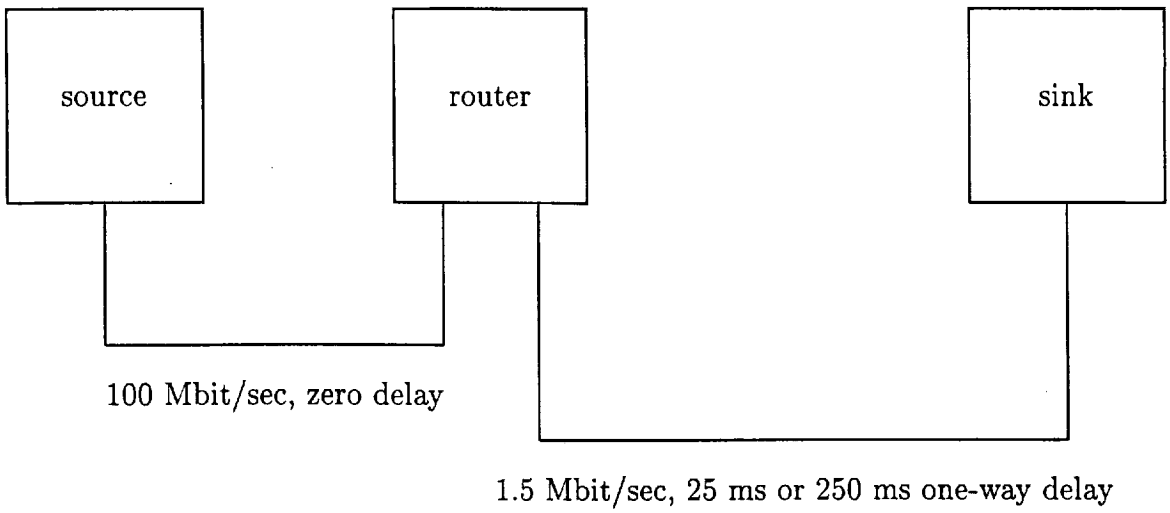


Figure 1: The topology we used for all simulations of the various TCPs available in the LBL NS simulator. The one-way delay on the long link for simulation of a "satellite" link was 250 ms, and for simulation of a "terrestrial" link was 25 ms. Traces of packets traveling in both directions were collected as they were transmitted on the 100 Mbit/sec no-delay link.

flow control window from the receiver, but just simulate a constant window from the highest acknowledged packet.

Despite these shortcomings, we chose to use the NS simulator because we felt we could get to work faster using it than using any other alternative of which we were aware.

5 Our first simulations

We modified existing sample simulations scripts of NS to produce output in the format of time-sequence plots like those developed in [5], and ran some initial simulations to demonstrate how an end-to-end flow control window that is too small can limit throughput, and that if the end-to-end flow-control window is opened up, then the congestion window can limit throughput.

Now we were ready to simulate, but were still seeking to understand and separate the issues involved. While thinking about that, we tried running the different TCP modules available in NS in different situations, and explored the resulting time sequence plots teaching myself how to explain why each packet was sent when it was. Our goal was to understand the fundamental problems (and then perhaps fix them), not to produce more anecdotal evidence of some things going slower and other things going faster.

Figure 1 shows the topology we used for the simulations in the LBL NS simulator. The long link was the bottleneck in all cases, and any packets dropped were dropped in the router due to a queue overflow. The 100 Mbit/sec link was included (instead of using a simpler topology) so that the returning packets carrying acknowledgements could be logged by the normal packet tracing mechanisms in NS.

See Figure 2 demonstrates the problem when the end-to-end flow-control window is too small. As can be seen more clearly in Figure 3, once the TCP transmitter has filled the window, it must wait a round-trip time before proceeding further.

Figure 4 and Figure 5 show plots from the same experiments, but with the end-to-end flow-control window opened wide enough so that it is not a factor. The throughput of the satellite case still suffers due to the congestion-related algorithms in the TCP transmitter which limit the number of packets that may be sent but not yet acknowledged.

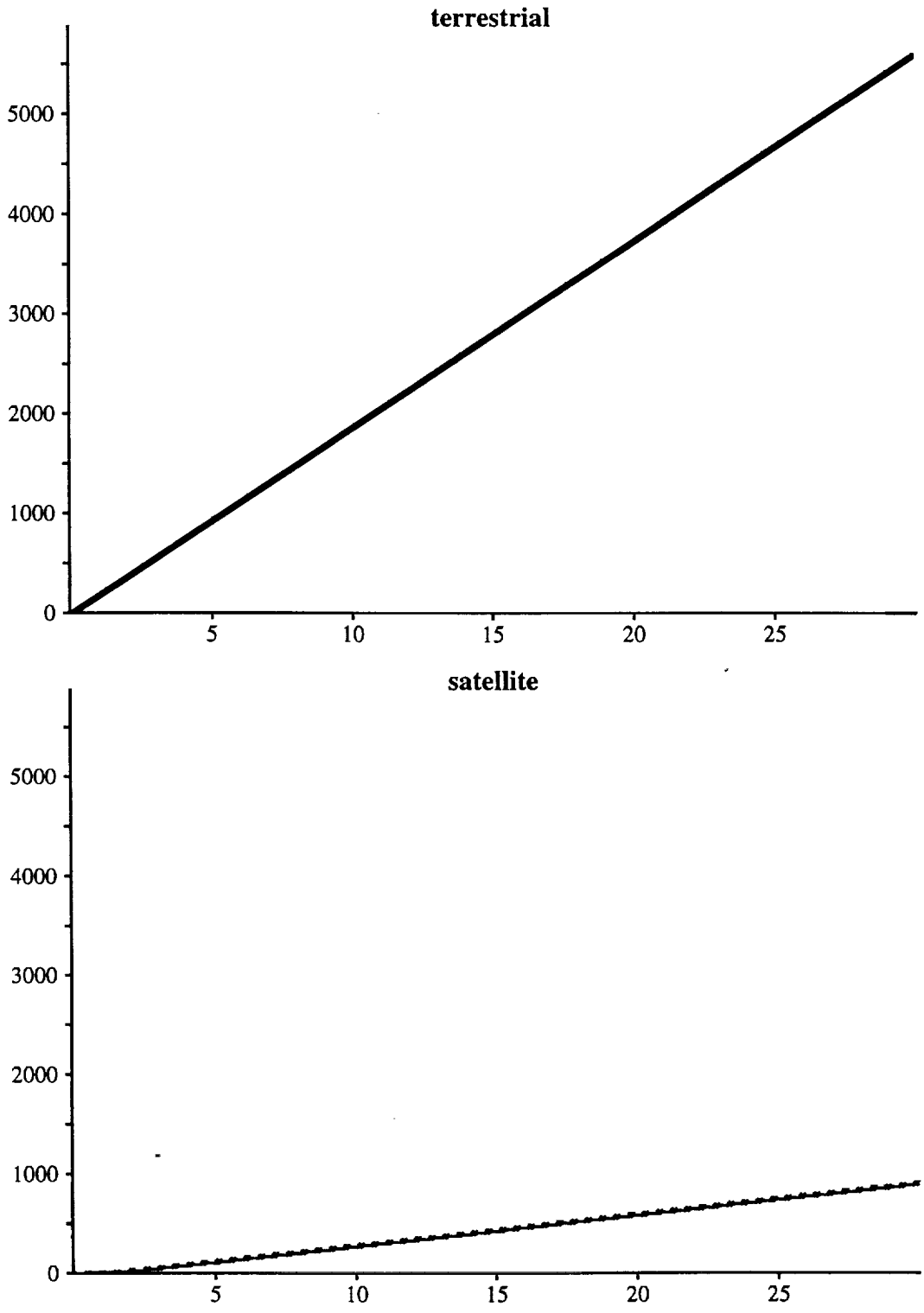


Figure 2: A comparison of the performance over a terrestrial link and a satellite link of a typical TCP which does not open wide the end-to-end flow control window. In the case of the satellite, the window is limiting the throughput severely. Figure 3 shows these same two plots at a magnified scale.

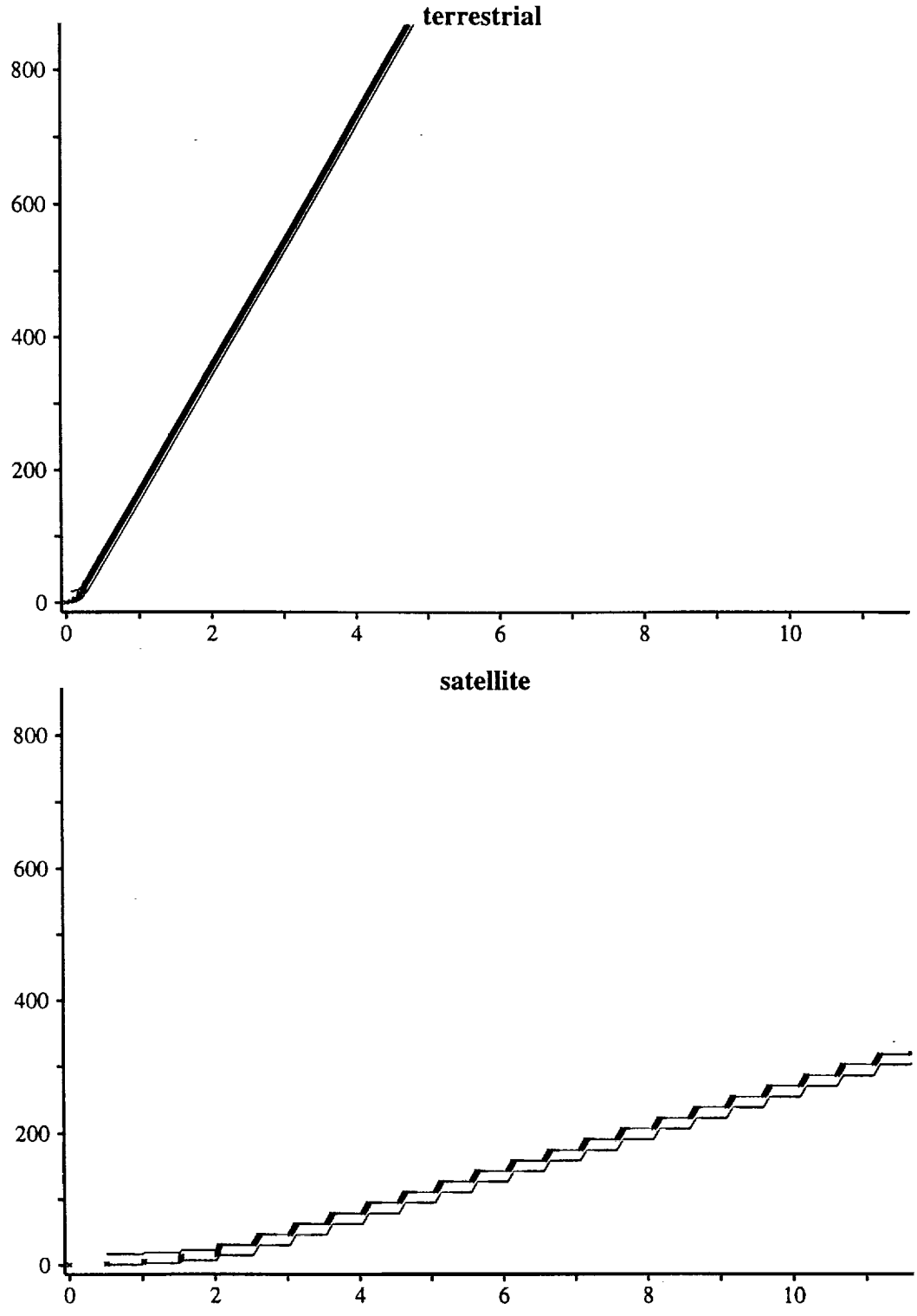


Figure 3: A closer view of the two plots in Figure 2.

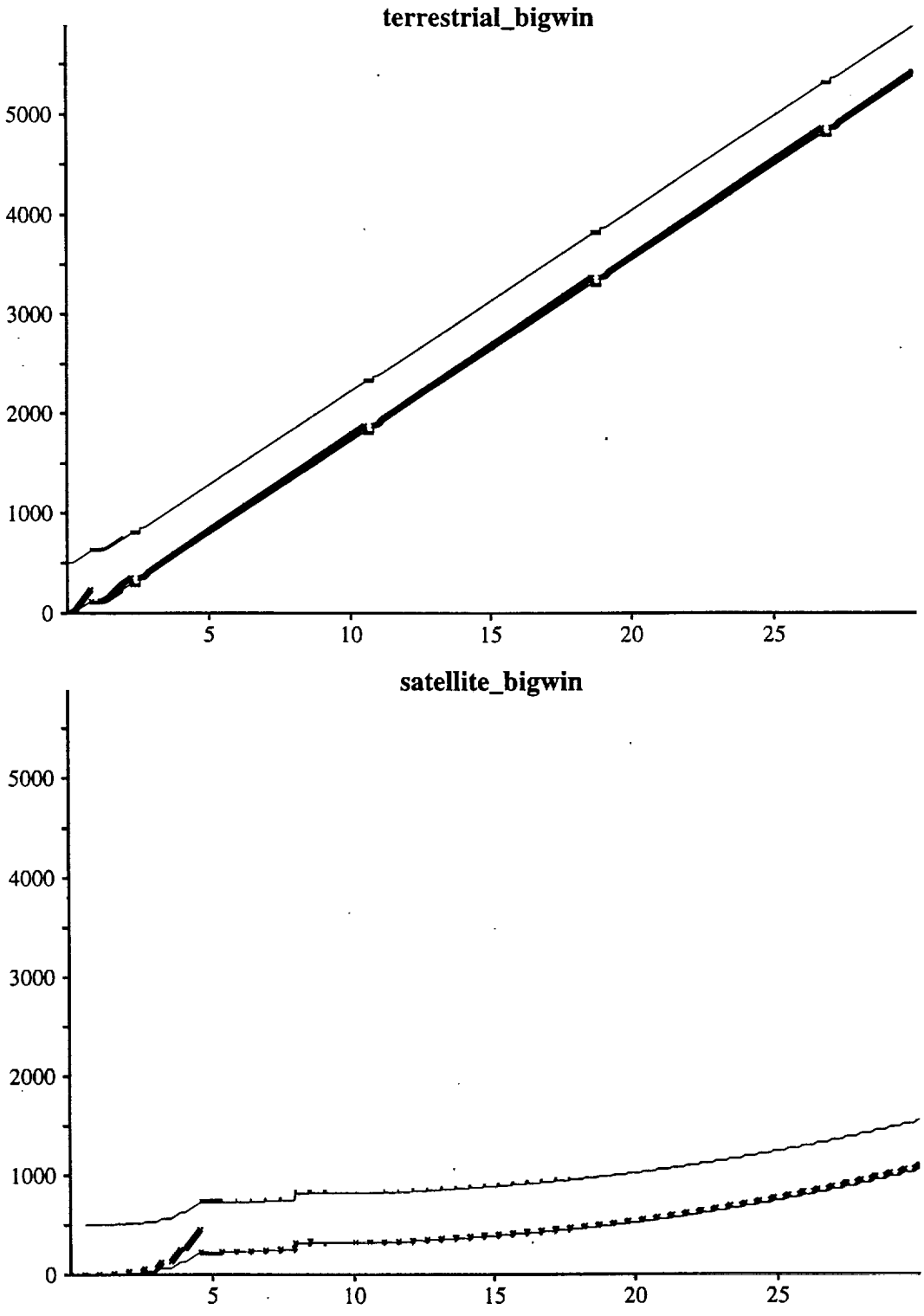


Figure 4: The same comparison (plotted at the same scale) as Figure 2, but with the end-to-end flow control window open wide enough so that it is not a factor. In both cases packet losses due to a queue overflow cause the congestion-related algorithms in TCP to limit the TCP transmitter to less than the offered window. In the terrestrial case, this does not limit throughput. But in the satellite case, throughput is limited. Figure 5 shows these same two plots at a magnified scale.

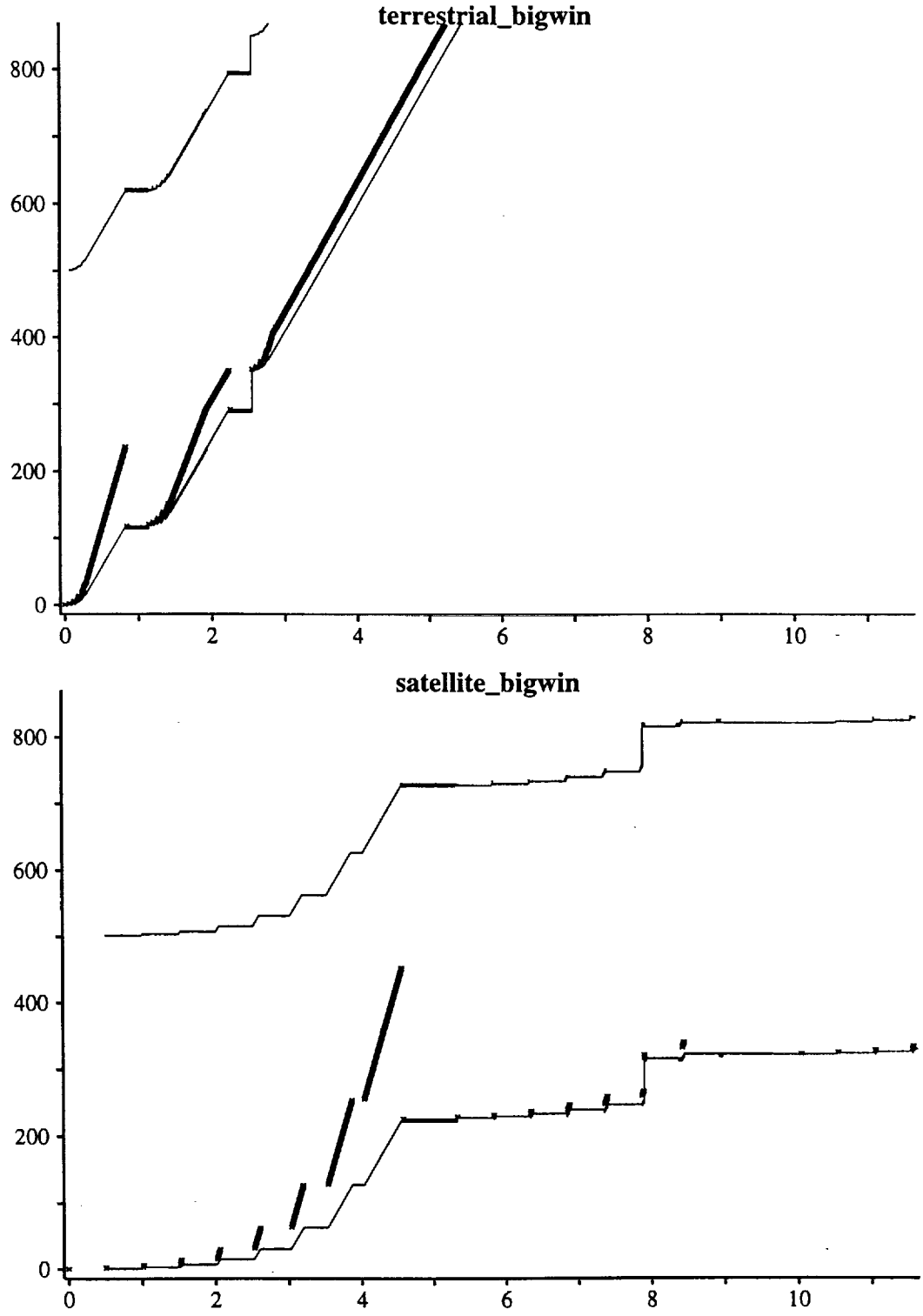


Figure 5: A close view of the two plots in Figure 4 (at the same scale as in Figure 3).

6 December 1996 IETF (San Jose)

Dr. Shepard traveled to the December IETF, and met up with others, most notably Aaron Falk (of TRW), who are interested in how the Internet protocols can be made to work over satellite communication systems. Aaron had been appointed chair of a TIA working group chartered to look into the problem of data protocols over satellite. Prior to the IETF Dr. Shepard had exchanged e-mail with him to help introduce him to the workings of the IETF. Aaron was trying to organize those who had interest in the subject to work together to try to make TCP (and any other protocols later developed within the IETF) work better over paths that include satellite links. At this December IETF, Dr. Shepard began an extensive discussion (which continued through the April IETF in Memphis) with Aaron to bring him up to speed on how the congestion-related algorithms in TCP are an issue and how they are an important architectural element of the Internet today. Dr. Glover (of NASA) was also involved in some of these discussions.

Dr. Shepard was invited to present a plenary talk at this IETF on the plotting method for TCP packet traces [5] that he had developed in the late 1980s. At the end of the plenary talk, he included a few plots from the simulations in Figures 2 thru 5 showing how a performance problem can arise when TCP is used over a T1-rate satellite link and showed how just getting the end-to-end window opened wide is not sufficient to solve the problem (because the congestion-related algorithms will limit how many packets can be sent). This was to an audience of approximately 1500 people, and helped to raise the awareness of this issue in the larger IETF community.

Our contribution at this point was to raise awareness that fixing the TCP-over-satellite performance problems will need to involve more than just deployment of the options described in RFC2018 (TCP Selective Acknowledgement Option) and RFC1323 (TCP Extensions for High Performance). These two documents were well on their way through the standards process in the IETF at this point, and it seemed that in some people's minds it appeared that these may be all that would be needed.

7 Cleveland Meeting of the TIA/SCD/CIS/IPWG

Aaron called for a meeting in late January of the Internet Protocols Working Group of the Communications Interoperability Section of the Satellite Communications Division of the TIA. This meeting was hosted by NASA Lewis Research Center in Cleveland. The purpose of the meeting was to bring the satellite industry folks together to help the satellite industry understand better what would need to be done to effectively carry the Internet protocols over satellite systems. At this meeting Dr. Shepard gave a talk explaining how TCP's congestion control algorithms work and showed plots of traces from simulation runs showing packet-by-packet what happens when the delay is large, and how there is a sensitivity to the amount of buffering available at the bottleneck.

This talk clearly indicated to those present that opening the end-to-end window (as the extensions in RFC1323 allow), while necessary, is not sufficient to achieve reasonable performance over a geosynchronous satellite link of moderate rate. We were just beginning to understand the issue regarding buffering at the bottleneck and delay-bitrate product, and while responding to a question, Dr. Shepard realized that this problem is not easily solved locally by the deployers of the satellite link because the bottleneck may be elsewhere in the network. The deployers of the satellite link may have no control over the amount of buffering available elsewhere in the network.

A different perception that some had was that a problem is that packets lost due to errors can drive the congestion control algorithms to inappropriate places. This is true. But what is not true is that preventing error losses solves this problem. Packets lost due to congestion can also drive the congestion control algorithms to inappropriate places, and because the delay is longer over satellite links, TCP's congestion algorithms can take much longer to recover, and poor throughput will occur while the recovery is underway.

8 Further simulation study

After the January meeting, we investigated the dependency on the amount of buffering available at the bottleneck by simulating each of the exiting TCP models available in the LBL NS simulator while varying the amount of buffering. In each of these studies, a single TCP was made to transfer bulk data over a

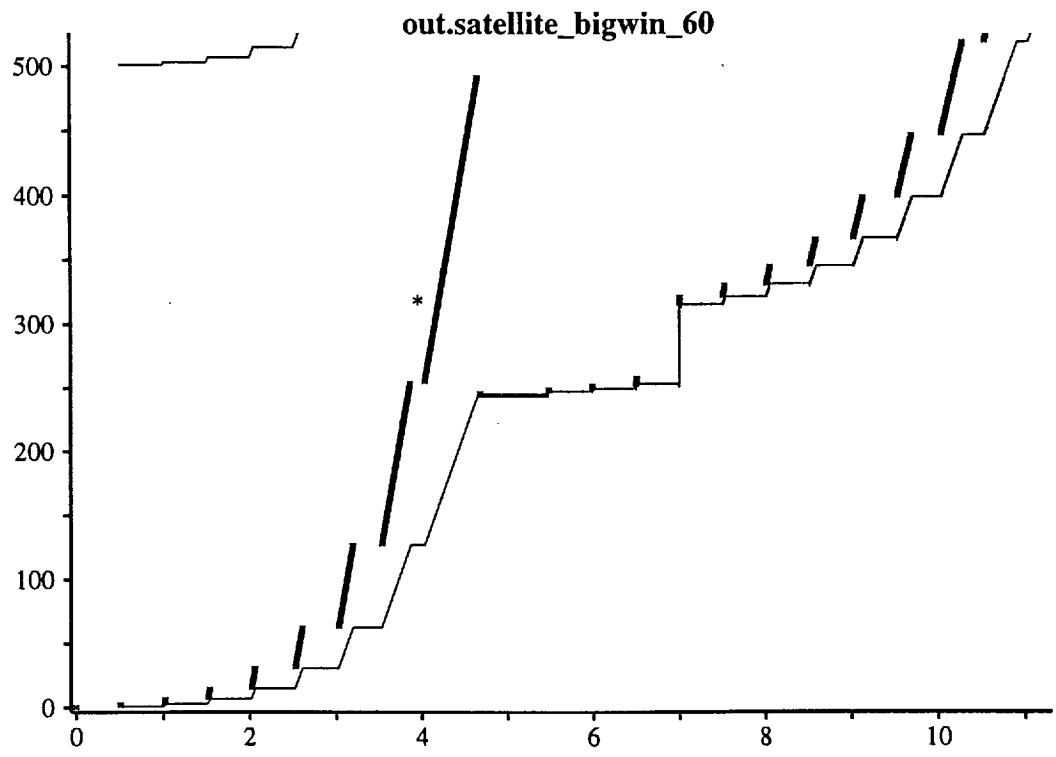


Figure 6: A plot illustrating where every other packet is dropped once the queue at the bottleneck is full. Starting with packets near the asterisk, every other packet is dropped at the queue at the bottleneck. Every other packet is dropped because they are arriving at the queue at precisely twice the rate at which packets are leaving the queue. Each packet that leaves the queue generates one returning packet to carry the acknowledgement. Each returning packet causes two new packets to be sent.

1.5 Mbit/second link with a round-trip time of one-half second. The delay-bitrate product for this path corresponds about 90 packets worth of data (assuming 1024 bytes per packet). The number of packet buffers available at the bottleneck was set to 6, 20, 60, and 200 on different runs. The end-to-end flow-control window was set large enough so that it would never be a factor.

Examination of the plots from these simulations taught us something about the slow-start algorithm. The slow-start algorithm is usually described as an exponential increase in the number of packets outstanding: one packet in the first round trip time, and then a doubling of the number of packets in each subsequent round-trip time. But this simple explanation (while accurate) neglects to mention that the round-trip time grows as the number of packets filling queues in the network increases.

The slow-start algorithm is more directly described as: start with one packet, then for each acknowledgement, increase by one the number packets we permit to be outstanding (sent but not yet acknowledged). The effect is to send two packets for each acknowledgement received (assuming the receiver is not dallying). Since packets exit the network at the rate of the bottleneck link, the effect of the slow-start algorithm is to send at *twice* the rate of the bottleneck link, until the sender learns of the first dropped packet. The result of this is that the queue at the bottleneck link grows at the same rate as packets go through the bottleneck link, and once full, every other packet is dropped (until the sender learns of the first drop, which will take a full round-trip time, which at this point is inflated by the full queue). Every other packet is dropped because each packet that leaves the queue generates an acknowledgement, which causes two packets to be sent by the sender. So for each new opening in the queue, two packets are sent, and only one of them can fit. Figure 6 shows where this happens.

These simulations also demonstrated the value of selective acknowledgements. The TCP models that did not incorporate the RFC2018 extension for selective acknowledgements all performed quite poorly while

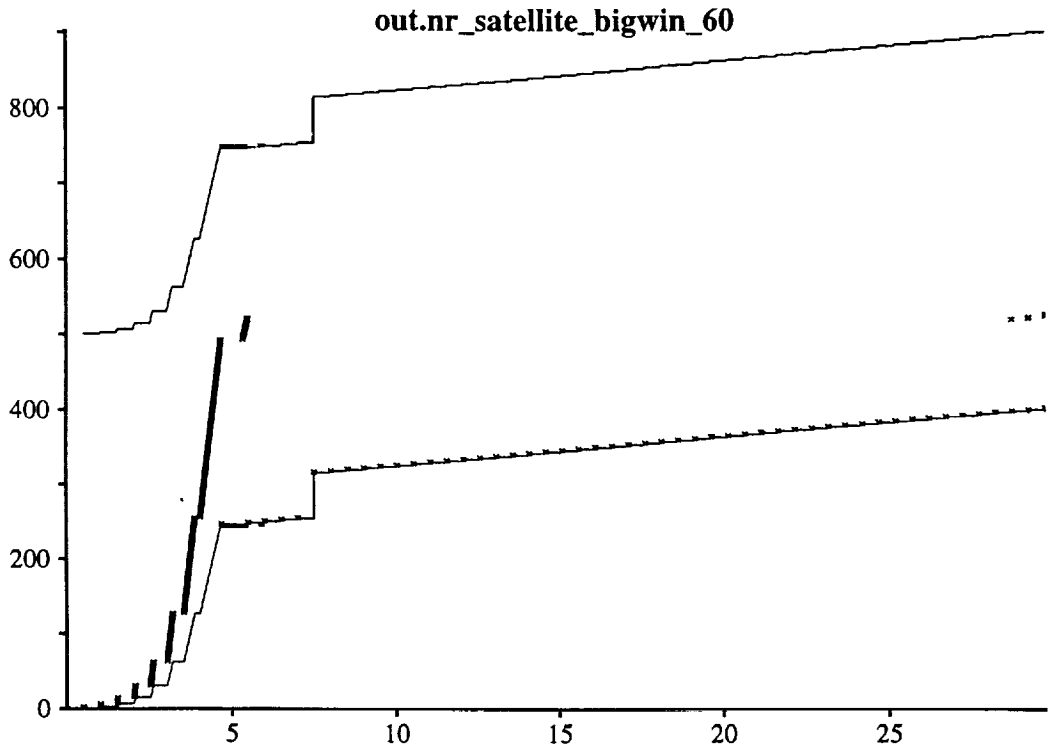


Figure 7: The behavior of the `newreno` TCP module in the LBL NS simulator. During the recovery after every other packet is lost at the end of the slow-start episode, it takes one full round trip to recover each packet.

recovering from this pessimal pattern of packet losses. In general, they either retransmit many packets unnecessarily (in Figure 6, most every packet is retransmitted, even though only every other packet was actually missing), or take one full round-trip time to recover each lost packet (which is shown in Figure 7). In the simulations of the two TCP models that do include use of selective acknowledgements, only the missing packets were retransmitted.

The behavior of the FACK TCP (one of the two which included use of selective acknowledgements) was particularly impressive. It manages to recover quickly, losing little or no time, and if the resulting congestion window does not limit throughput, data is delivered to the receiver at the full rate of the bottleneck link with no gap in performance due to the packets lost (even though a full delay-bitrate product in packets were lost). With FACK and sufficient buffering at the bottleneck, the the link never goes idle and never carries any unnecessary retransmissions. So full use of the bottleneck link is achieved. This can be seen in Figure 8.

To further study the consequences of varying the amount of buffering at the bottleneck link, the FACK TCP was simulated over the same topology, but with amounts of buffering at the bottleneck link corresponding to 10, 20, 30, 40, 50, 60, ..., 190, and 200 packets. These simulations confirmed the well-known rule-of-thumb that a router should have enough buffer space to hold one round-trip-time's worth of traffic. Below 90 packets of buffering, performance suffered due to the congestion window being reduced to one-half of the value it had when the sender first learned of a loss. With sufficient buffer space at the bottleneck, the resulting congestion window (after the loss was known to the sender and the initial slow-start episode ends) was still large enough to make full use of the available bandwidth.

9 Memphis IETF

At the Memphis IETF, we participated in the TCPSAT BOF, a session called to discuss forming a group within the IETF to document the problems with, and any existing methods of improving performance of,

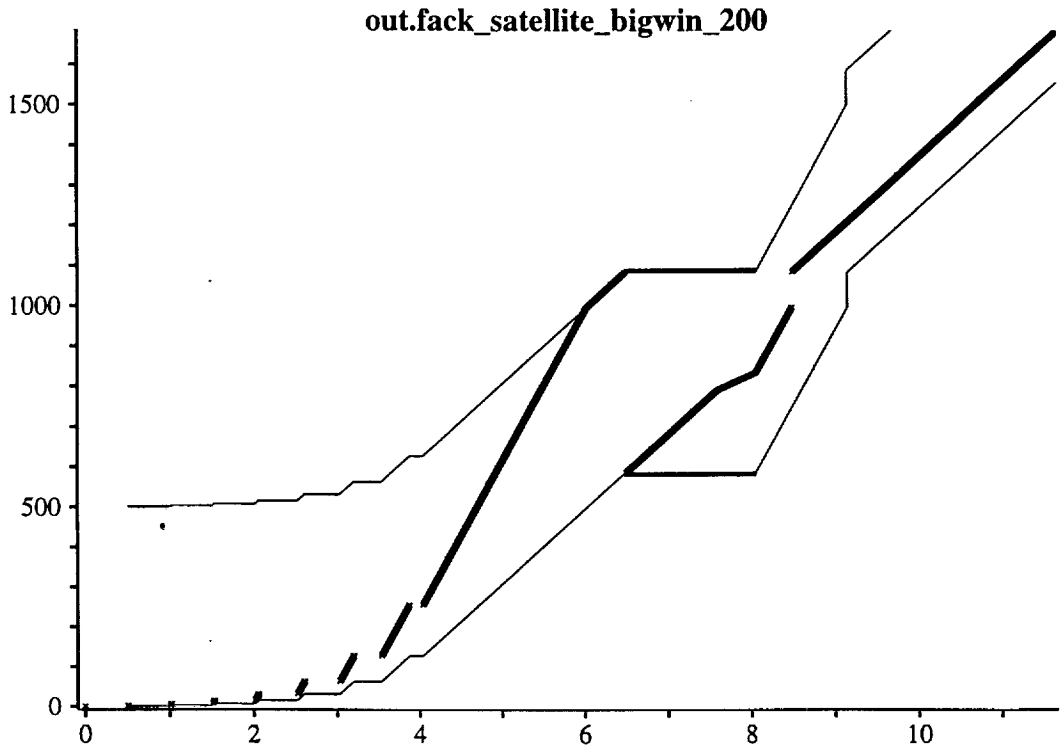


Figure 8: The behavior of the `fack` TCP module in the LBL NS simulator. The FACK algorithm manages to recover in this case from the pessimal pattern of lossage at the end of the slow-start episode without falling behind in performance at all. This can be seen where after the recovery, the returning ACKs fall on the same line as the ACKs before the period of recovery.

TCP over satellites. Dr. Shepard spoke at this meeting and explained the problem with the congestion-related algorithms. He also pointed out that for the receiver side, we now know what we need to do (RFC1323, RFC2018, and open the window wide). But that for the sender side, more research may be needed. This is good news, because we can without doing any more research encourage implementors to include the RFC1323 and RFC2018 extensions and also open the TCP receive windows as wide as possible.

10 Four-packet start

After the Memphis IETF, we did a quick study of the effects of starting the initial TCP slow-start algorithm with four packets instead of one. An Internet Draft proposing a four-packet start from Sally Floyd had already been distributed informally, and many people were discussing it. The motivation for this is that if it takes a fixed number of round-trips to perform an operation (which is usually the case for TCP transfers), and the round-trip delay is increased (perhaps because a satellite link has been introduced), then we should strive to complete the operation in as few round trip times as possible. In the first two round-trip times of a normal TCP slow-start, only three packets are transferred, and the TCP has very little chance to learn anything useful about the congestion-related characteristics of the path (other than if it is working or not). So the proposal was to begin TCP transfers with a congestion window which would allow four packets to be launched.

Concerns had been raised that this four-packet-start change may cause trouble for TCPs sending to users who are dialed into modem pools (which may have as few as three buffers). My study was limited to determining the consequences of starting with four packets over a path with a slow modem at then end and with only three packet buffers (which can queue only two packets while another single packet is being transmitted).

The results of this study were written up in an Internet Draft (ID) (copy attached). This ID explains that while some packets are delivered later with a 4-packet start, there is long-term effect on throughput or latency. Identical patterns (modulo a shift in time) of packet transmission and loss occur in the two cases (a one-packet or four-packet start).

11 Testing real FACK TCP

We were concerned that the simulations in NS simulator may be missing some important issue, since the TCP models in the NS simulator are only brief approximations of a real TCP implementation. An experimental implementation of FACK TCP was available from the Networking Research Group at the Pittsburgh Supercomputing Center, and we were also eager to see the FACK algorithms in action in a real TCP. This motivated the construction of a small (two machine) test bed. The the test bed consisted of two machines running NetBSD, with the FACK TCP replacing the regular TCP in both systems. The DUMMYNET delay simulator [4] was included to simulate delays, link speeds, and queue limits.

A few test runs on the test bed confirmed our findings of the queue-overrun behavior of the slow-start algorithm. Once the queue is full, every other packet is dropped until the sender learns of the first dropped packet (which takes one RTT or more).

The FACK TCP does in fact perform remarkably well. It recovers from the loss by resending only the packets which need to be resent, and it keeps the bottleneck link active at all times (so it never sits idle wasting time). In the simple topologies we tested, it would be impossible to improve upon the perceived performance of FACK TCP for transfers of long files. It may be possible to reduce the number of packets sent needlessly (i.e. those sent but dropped before the bottleneck) but that would not improve upon the throughput or latency seen by the user of the FACK TCP. We believe that TCPs employing the FACK algorithms should be the benchmark against which performance is measured when testing the performance of TCP over satellite links.

This experimental implementation of FACK TCP (from the PSC) does have a few problems which should be addressed before advocating its widespread use. The first is a bug (which perhaps has nothing to do with FACK or SACK) which causes delayed acknowledgements to perform incorrectly. (In our tests we patched around this bug by disabling delayed ACKs.) The other bug (or bugs?) is that the FACK TCP appears to cause unreliability of the system. A NetBSD workstation in regular daily use with this experimental FACK TCP installed crashes a few times per week, apparently due to a problem in TCP.

12 Queue overrun observations conveyed to End-to-End Research Group

The thoughts resulting from our study of the slow-start algorithm's overrunning of the buffers at the bottleneck link were written up and conveyed in a private e-mail message to Craig Partridge prior to a meeting of the End-to-End research group meeting in the summer. A very-slightly edited copy of that message is included (attached).

Dr. Partridge wrote an Internet Draft to capture an idea discussed at an End-to-End Research Group meeting. The idea is that perhaps a box inserted in the network near the satellite link could cause the source to pace its transmissions by pacing the ACKs that are being returned to it. A copy of this draft is included (attached).

13 IETF Munich

We attended the IETF in Munich in August 1997. A number of members of the End-to-End Research Group attended the TCP Over Satellite Working Group meeting. Dominating the meeting was a discussion about the need for pacing to avoid the queue-overrun problem caused by the slow-start algorithm.

14 IEEE Network publication

A paper describing performance issues involved with TCP and IP over satellites links was written and appears in the October 1997 issue of IEEE Network. Dr. Glover requested we write and publish this to help clarify what is and is not a problem with the IP and TCP protocols when long delays are involved. A copy of this publication is included (attached).

15 Conclusion and Future Work

In the first year of this project, we believe we have uncovered the need for pacing in TCP senders to avoid a problem where the slow-start algorithm is likely to overrun a queue at the bottleneck link. Exactly how this pacing should work is unclear at this time. Further thought, implementation, and experimentation will be needed.

References

[1] R. Braden. Requirements for internet hosts — communication layers. Request for Comments 1122, DDN Network Information Center, SRI International, October 1989.

[2] V. G. Cerf. Protocols for interconnected packet networks. *Computer Communication Review*, 10(4), October 1980.

[3] Van Jacobson. Congestion avoidance and control. In *Proceedings of SIGCOMM '88*, August 1988.

[4] L. Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *Computer Communication Review*, 27(1), January 1997.

[5] Timothy J. Shepard. Tcp packet trace analysis. Technical Report LCS/TR/494, Massachusetts Institute of Technology Laboratory for Computer Science, Cambridge, MA, January 1991. on the Internet at `ftp://ftp-pubs.lcs.mit.edu/pub/lcs-pubs/tr.outbox/MIT-LCS-TR-494.ps.gz`.

Attached are four items:

- The contents of an e-mail message sent from Dr. Shepard to Dr. Partridge immediately prior to Dr. Partridge attending an End-to-End Research Group meeting in July of 1997. (This message is included here because it represents well our thinking.)
- An Internet Draft we submitted on the 4-packet-start.
- An Internet Draft we submitted on ACK spacing.
- A paper on TCP/IP performance over satellites we published in IEEE Network.

From: Tim Shepard <shep@bbn.com>
To: Craig Partridge <craig@aland.bbn.com>
Subject: pacing
Date: Mon, 07 Jul 1997 15:25:29 -0400
Sender: shep@marengo.bbn.com

Craig,

[...]

I have looked at the TCPs in the LBL simulator, and examined their behavior in situations where the delay-bitrate product is 90 packets. (Remember that the simulator simulates just packets, not TCP.) SACK reveals something I never understood previously. The slow start algorithm sends at twice the rate of the bottlneck, so once the queue overflows, every other packet gets dropped (for a full RTT time). For example, with a 90-packet delay-bitrate product, and 90 packets of buffering at the bottleneck, 90 packets will be dropped out of 180 packets sent in the round trip time immediately proceeding the arrival of the first duplicate acks (which are the first notification to the sender that something is amiss.)

The FACK tcp (in the LBL simulator) does a remarkable job of recovering from this pessimal lossage without losing any ground.
[...]

So what is the problem that a satellite link in the net adds? It adds a link with longer-than-usual delay, increasing the delay-bandwidth product. Hence need large windows. (Need window scaling, in BSD TCP need to increase socket buffer size at both sender and receiver, probably need SACK, 4k-start is probably a good idea, etc.)

But if buffering at bottleneck is significantly less than a delay-bitrate product, slow-start in its current form is incapable of getting ssthresh set to a reasonable value because by sending at twice the rate of the bottleneck, it will overrun the queue.

*** Slow start learns the cwnd size it must not exceed if it wants to transmit at twice the rate of the bottleneck without overflowing a queue. ***

If you want to learn the window size you should use to send at the rate of the bottleneck, then you need to "tip over" and send at the rate of the bottleneck. (With perhaps a handful of extra packets in the net to ensure that you are pushing hard enough to learn something.

When I explained this to DDC, he pointed out that if you set ssthresh correctly in the first place, then you'll get just the "tip over" effect that I want.

I said, "but not if there's not enough buffering," thinking that it required a full delay-bitrate product's worth at the bottleneck.

But I was off by a factor of two. If the bottlneck has at least one-half the delay-bitrate product in buffering, then if you somehow get ssthresh set correctly (((perhaps using Janey Hoe's method if it should prove robust enough (((but my guess is that something which made use of more data from the returning acks would be better)))))), slow-start can get you going up to full speed in "just a few" round trips.

That is so because if you set ssthresh correctly, then at the point you tip over, you will have put in flight a full delay-bitrate product's worth of packets, but in one half of a round-trip time. In that time, half will have drained from the queue, and the other half will be sitting in

the queue filling it up.

(((Note, with ssthresh set to plus infinity at the start, then in order to win at the end of the slow-start episode, you need to have a full delay-bitrate product's worth of buffering so that from the point of view of the sender, it can get to a cwnd which is *twice* the delay-bitrate product before it first learns of the loss. In the ninety-packet delay-bitrate product case (T1-line, 1/2 second round trip time, 1024-byte packets) you have to get to 180 packets outstanding, half of which will be in-flight and half of which are sitting in the queue, so that when ssthresh is set to 1/2 cwnd, it gets a value of 90 packets.)))

But there is still a problem, if the buffering available at the bottleneck router is less than one half of the delay-bitrate product experienced by a connection, then even with ssthresh set correctly, the queue will be overrun before the sender has managed to put a full delay-bitrate product's worth into the net. (Because the sender is sending at twice the rate of the bottlneck link, it overruns a queue that it would not overrun at the proper rate.)

It is unreasonable to expect that the party responsible for maintaining the router at the bottleneck to have any awareness that some of of the traffic through it is experiencing long delays because of long-delay links elsewhere in the net.

Hence probing should be done at the rate of returning acks, not at twice the rate of the returning acks. This will require some amount of pacing timers to pace out the packets, but not many pacing timer interrupts need be taken by the operating system. Incoming acks can still be used to pace out a majority of slow-start packets. (I appeal to a visual argument on my pencel-marked-up plot at this point.) Pacing timers need not be per-packet either. It'd probably suffice to release a few packets each time the pacing timer goes off.

Pacing timers should not be viewed as overly burdensom for an operating system in 1997. In an extreme example, imagine the rate of the bottleneck is 600 million bits per second. That's 75 million bytes per second, or fifty-thousand 1500-byte packets per second. If a pacing timer released five packets on each firing, then only 10,000 pacing-timer interrupts per second need be taken. That's once every 100 micro-seconds. 10,000 interrupts per second on a 100 MHz Pentium is like taking 100 interrupts per second on a VAX-11/780. On any machine capable of noticing a 600 megabit-per-second bottlneck in the net, 10,000 interrupts per second should not be a big deal.

[...]

-Tim

Tim Shepard
Craig Partridge
BBN Technologies
July 1997

This Internet Draft expires February 4, 1998.

When TCP Starts Up With Four Packets Into Only Three Buffers

Status of this Memo

Background and Abstract

Sally Floyd has proposed that TCPs start their initial slow start by sending as many as four packets (instead of the usual one packet) as a means of getting TCP up-to-speed faster. (Slow starts instigated due to timeouts would still start with just one packet.) Starting with more than one packet might reduce the start-up latency over long-fat pipes by two round-trip times. This proposal is documented further in [1] and in [2] and we assume the reader is familiar with the details of this proposal.

On the end2end-interest mailing list, concern was raised that in the (allegedly common) case where a slow modem is served by a router which only allocates three buffers per modem (one buffer being

transmitted while two packets are waiting), that starting with four packets would not be good because the fourth packet is sure to be dropped.

Vern Paxson replied with the comment (among other things) that the four-packet start is no worse than what happens after two round trip times in normal slow start, hence no new problem is introduced by starting with as many as four packets. If there is a problem with a four-packet start, then the problem already exists in a normal slow-start startup after two round trip times when the slow-start algorithm will release into the net four closely spaced packets.

This memo is to document that in the case of a 9600 bps modem at the edges of a fast Internet where there are only 3 buffers before the modem (and the fourth packet of a four-packet start will surely be dropped), no significant degradation in performance is experienced with a four-packet start when compared with a normal slow start (which starts with one packet).

Scenario and experimental setup

The scenario studied and simulated consists of three links between the source and sink. The first link is a 100 Mbps link with no delay. (It was included to have a means of logging the returning ACKs at the time they would be seen by the sender.) The second link is a 1.5 Mbps link with a 25 ms one-way delay. The third link is a 9600 bps link with a 150 ms one-way delay. The queue limits for the queues at each end of the first two links were set to 100 (a value sufficiently large that this limit was never a factor). The queue limits at each end of the 9600 bps link were set to 3 packets (which can hold at most two packets while one is being sent).

Version 1.2a2 of the the NS simulator (available from LBL) was used to simulate both one-packet and four-packet starts for each of the available TCP alogorithms (tahoe, reno, sack, fack) and the conclusion reported here is independent of which TCP algorithm is used (in general, we believe). The "tahoe" module will be used to illustrate what happens in this memo. In the 4-packet start cases, the "window-init" variable was set to 4, and the TCP implementations were modified to use the value of the window-init variable only on connection start, but to set cwnd to 1 on other instances of a slow-start. (The tcp.cc module as shipped with ns-1.2a2 would use the window-init value in all cases.)

The packets in simulation are 1024 bytes long for purposes of determining the time it takes to transmit them through the links. (The TCP modules included with the LBL NS simulator do not simulate the TCP sequence number mechanisms. They use just packet numbers.)

Observations are made of all packets and acknowledgements crossing the 100 Mbps no-delay link. (All descriptions below are from this point of view.)

What happens with normal slow start

At time 0.0 packet number 1 is sent.

At time 1.222 an ack is received covering packet number 1, and packets 2 and 3 are sent.

At time 2.444 an ack is received covering packet number 2, and packets 4 and 5 are sent.

At time 3.278 an ack is received covering packet number 3, and packets 6 and 7 are sent.

At time 4.111 an ack is received covering packet number 4, and packets 8 and 9 are sent.

At time 4.944 an ack is received covering packet number 5, and packets 10 and 11 are sent.

At time 5.778 an ack is received covering packet number 6, and packets 12 and 13 are sent.

At time 6.111 a duplicate ack is recieved (covering packet number 6).

At time 7.444 another duplicate ack is received (covering packet number 6).

At time 8.278 a third duplicate ack is received (covering packet number 6) and packet number 7 is retransmitted.

(And the trace continues...)

What happens with a four-packet start

At time 0.0, packets 1, 2, 3, and 4 are sent.

At time 1.222 an ack is received covering packet number 1, and packets 5 and 6 are sent.

At time 2.055 an ack is received covering packet number 2, and packets 7 and 8 are sent.

At time 2.889 an ack is received covering packet number 3, and packets 9 and 10 are sent.

At time 3.722 a duplicate ack is received (covering packet number 3).

At time 4.555 another duplicate ack is received (covering packet number 3).

At time 5.389 a third duplicate ack is received (covering packet number 3) and packet number 4 is retransmitted.

(And the trace continues...)

Discussion

At the point left off in the two traces above, the two different systems are in almost identical states. The two traces from that point on are almost the same, modulo a shift in time of (8.278 - 5.389) = 2.889 seconds and a shift of three packets. If the normal TCP (with the one-packet start) will deliver packet N at time T, then the TCP with the four-packet start will deliver packet N - 3 at time T - 2.889 (seconds).

Note that the time to send three 1024-byte TCP segments through a 9600 bps modem is 2.66 seconds. So at what time does the four-packet-start TCP deliver packet N? At time T - 2.889 + 2.66 = T - 0.229 in most cases, and in some cases earlier, in some cases later, because different packets (by number) experience loss in the two traces.

Thus the four-packet-start TCP is in some sense 0.229 seconds (or about one fifth of a packet) ahead of where the one-packet-start TCP would be. (This is due to the extra time the modem sits idle while waiting for the dally timer to go off in the receiver in the case of the one-packet-start TCP.)

The states of the two systems are not exactly identical. They differ slightly in the round-trip-time estimators because the behavior at the start is not identical. (The observed round trip times may differ by a small amount due to dally timers and due to that the one-packet start experiences more round trip times before the first loss.) In the cases where a retransmit timer did later go off, the additional difference in timing was much smaller than the 0.229 second difference discribed above.

Conclusion

In this particular case, the four-packet start is not harmful.

Non-conclusions, opinions, and future work

A four-packet start would be very helpful in situations where a long-delay link is involved (as it would reduce transfer times for moderately-sized transfers by as much as two round-trip times). But it remains (in the authors' opinions at this time) an open question whether or not the four-packet start would be safe for the network.

It would be nice to see if this result could be duplicated with real TCPs, real modems, and real three-buffer limits.

References

1. S. Floyd, Increasing TCP's Initial Window (January 29, 1997). URL ftp://ftp.ee.lbl.gov/papers/draft.jan29.

2. S. Floyd and M. Allman, Increasing TCP's Initial Window (July, 1997). URL http://gigahertz.lerc.nasa.gov/~mallman/share/draft-ss.txt (To be submitted as an Internet Draft).

Authors' Addresses:

Tim Shepard, Craig Partridge
BBN Technologies
10 Moulton Street
Cambridge, MA 02138

shep@bbn.com, craig@bbn.com

End-To-End Research Group
Request for Comments: DRAFT
Category: Informational

C. Partridge
BBN Technologies
July 1997

ACK Spacing for High Delay-Bandwidth Paths with Insufficient Buffering

<draft-partridge-e2e-ackspacing-00.txt>

Status of this Memo

1. Introduction

Suppose you want TCP implementations to be able to fill a 155 Mb/s path. Further suppose that the path includes a satellite in a geosynchronous orbit, so the round trip delay through the path is at least 500 ms, and the delay-bandwidth product is 9.7 megabytes or more.

If we further assume the TCP implementations support TCP Large Windows and PAWS (many do), so they can manage 9.7 MB TCP window, then we can be sure the TCP will eventually start sending at full

path rate (unless the satellite channel is very lossy). But it may take a long time to get the TCP up to full speed.

One (of several) possible causes of the delay is a shortage of buffering in routers. To understand this particular problem, consider the following idealized behavior of TCP during slow start. During slow start, for every segment ACKed, the sender transmits two new segments. In effect, this behavior means the sender is transmitting at *twice* the data rate of the segments being ACKed. And keep in mind the separation between ACKs represents (in an ideal world) the rate segments can flow through the bottleneck router in the path. So the sender is bursting data at twice the bottleneck rate, and a queue must be forming during the burst. In the simplest case, the queue is entirely at the bottleneck router, and at the end of the burst, the queue is storing half the data in the burst. (Why half? During the burst, we transmitted at twice the bottleneck rate. Suppose it takes one time unit to send a segment on the bottlenecked link. During the burst the bottleneck will receive two segments in every time unit, but only be able to transmit one segment. The result is a net of one new segment queued every time unit, for the life of the burst.)

TCP will end the slow start phase in response to the first lost datagram. Assuming good quality transmission links, the first lost datagram will be lost because the bottleneck queue overflowed. We'd like that loss to occur in the round-trip after the slow start congestion window has reached the delay-bandwidth product. Now consider the buffering required in the bottleneck link during the next to last round trip. The sender will send an entire delay-bandwidth worth of data in one-half a round-trip time (because it sends at twice the channel rate). So for half the round-trip time, the bottleneck router is in the mode of forwarding one segment while receiving two. (For the second half of the round-trip, the router is draining its queue). That means, to avoid losing any segments, the router must have buffering equal to half the delay-bandwidth product, or nearly 5 MB.

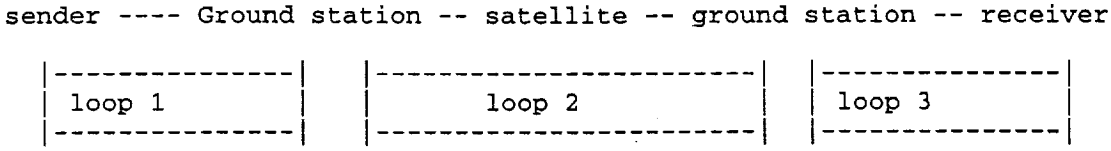
Most routers do not have anywhere near 5 MB of buffering for a single link. Or, to express this problem another way, because routers do not have this much buffering, the slow start stage will end prematurely, when router buffering is exhausted. The consequence of ending slow start prematurely is severe. At the end of slow start, TCP goes into congestion avoidance, in which the window size is increased much more slowly. So even though the channel is free, because we did not have enough router buffering, we will transmit slowly for a period of time (until the more conservative congestion avoidance algorithm sends enough data to fill the channel).

2. What to Do?

So how to get around the shortage of router buffering?

2.1 Cascading TCPs

One approach is to use cascading TCPs, in which we build a custom TCP for the satellite (or bottleneck) link and insert it between the sender's and receiver's TCPs, as shown below:



This approach can work but is awkward. Among its limitations are: the buffering problem remains (at points of bandwidth mismatches, queues will form); the scheme violates end-to-end semantics of TCP (the sender will get ACKs for data that has not and may never reach the receiver); and it doesn't work with encryption (i.e. if data above the IP layer is encrypted).

2.2 ACK Spacing

Another approach is to find some way to spread the bursts, either by having the sender spread out the segments, or having the network arrange for the ACKs to arrive at the sender with a two segment spacing (or larger).

Changing the sender is feasible, although it requires very good operating system timers. But it has the disadvantage that only upgraded senders get the performance improvement.

Finding a way for the network to space the ACKs would allow TCP senders to transmit at the right rate, without modification. Furthermore, it can be done by a router. The router simply has to snoop the returning TCP ACKs and spread them out. (Note that if the transmissions are encrypted, in many scenarios the router can still figure out which segments are likely TCP ACKs and spread them out).

There are some difficult issues with this approach. The most notable ones are:

1. What algorithm to use to determine the proper ACK spacing.

2. Related to (1), it may be necessary to known when a TCP is in slow-start vs. congestion-avoidance, as the desired spacing between ACKs is likely to be different in the two phases.

3. What to do about assymetric routes (if anything). If the ACKs do not return through the ACK-spacing router, it may not be possible to do ACK spacing.

Despite these challenges the approach has appeal. Changing software in a few routers (particularly those at likely bottleneck links) on high delay-bandwidth paths could give a performance boost to lots of TCP connections.

Credit and Disclaimer

This memo presents thoughts from a discussion held at the recent meeting of the End-To-End (E2E) Research Group. The particular idea of ACK spacing was developed by during the meeting by Mark Handley and Van Jacobson in response to an issue raised by the author, and was inspired, in part by ideas to enhance wireless routers to improve TCP performance [1].

The material presented is a half-baked suggestion and should not be interpreted as an official recommendation of the Research Group.

References

1. H. Balakrishnan, V.N. Padmanabhan, S. Seshan and R.H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links", Proc. ACM SIGCOMM '96, pp. 256-269.

TCP/IP Performance over Satellite Links

Craig Partridge and Timothy J. Shepard
BBN Technologies

Reprinted from
IEEE NETWORK
September/October 1997

TCP/IP Performance over Satellite Links

Craig Partridge and Timothy J. Shepard
BBN Technologies

Abstract

Achieving high data rates using TCP/IP over satellite networks can be difficult. This article explains some of the reasons TCP/IP has difficulty with satellite links. We present solutions to some problems, and describe the state of the research on some of the unsolved problems.

O of TCP/IP impact performance. We then present issues specific to satellites aninformal) about how well TCP/IP performs over satellite links. Some reports indicate TCP/IP throughput is poor. Others report that TCP/IP throughput is quite good. It is very difficult to determine which reports deserve more credence.

This article tries to clarify the situation. Our approach is to first discuss TCP/IP performance analytically, indicating what features of TCP/IP impact performance. We then present issues specific to satellites and their solutions, if known.

An Overview of TCP and IP Performance

TCP/IP is a surprising complex protocol suite and more than one person has written an entire book on the details of its operation.[1] Rather than try to summarize all of TCP/IP, our goal in this section is to present those aspects of TCP/IP that most directly affect TCP/IP throughput. More specifically, we will focus on a particular aspect of throughput, namely the effective transmission rate of valid data (sometimes called goodput) that a TCP/IP connection can achieve.

IP Throughput Issues

IP (the Internet Protocol) is the network layer protocol in the TCP/IP protocol suite. IP's function is to provide a protocol to integrate heterogeneous networks together. In brief, a media-specific way to encapsulate IP datagrams is defined for each media (e.g., satellite, Ethernet, or Asynchronous Transfer Mode). Devices called *routers* move IP datagrams between the different media and their encapsulations. Routers pass IP datagrams between different media according to routing information in the IP datagram. This mesh of different media interconnected by routers forms an IP *internet*, in which all hosts on the integrated mesh can communicate with each other using IP.[2]

The actual service IP implements is unreliable datagram delivery. IP simply promises to make a reasonable effort to deliver every datagram to its destination. However IP is free to occasionally lose datagrams, deliver datagrams with errors in them, and duplicate and reorder datagrams.

Because IP provides such a simple service, one might assume that IP places no limits on throughput. Broadly speaking, this assumption is correct. IP places no constraints on how fast a system can generate or receive datagrams. A system transmits IP datagrams as fast as it can generate them. However, IP does have two features that can affect throughput: the IP Time to Live and IP Fragmentation.

IP Time To Live — In certain situations, IP datagrams may loop among a set of routers. These loops are sometimes transient (a datagram may loop for a while and then proceed to its destination) or long-lived. To protect against datagrams circulating semipermanently, IP places a limit on how long a datagram may live in the network.

The limit is imposed by a Time To Live (TTL) field in the IP datagram. The field is decremented at least once at every router the datagram encounters and when the TTL reaches zero, the datagram is discarded.

Originally, the IP specification also required that the TTL also be decremented at least once per second. Since the TTL field is 8-bits wide, this means a datagram could live for approximately 4.25 minutes. In practice, the injunction to decrement the TTL once a second is ignored, but, perversely, specifications for higher layer protocols like TCP usually assume that the maximum time a datagram can live in the network is only two minutes.

This work was funded by NASA Lewis Research Center.

[1] Two very good books on the subject are [1] and [2].

[2] The term internet is a generic word for a group of interconnected networks. The Internet is the global IP internet. Recently the term intranet has evolved from its original meaning (an adjective meaning on a single physical network [3]) into a popular way to describe an IP internet entirely within an organization.

The significance of the maximum datagram lifetime is that it means higher layer protocols must be careful not to send two similar datagrams (in particular, two datagrams which could be confused for each other) within a few minutes of each other. This limitation is particularly important for sequence numbers. If a higher layer protocol numbers its datagrams, it must ensure that it does not generate two datagrams with the same sequence number within a few minutes of each other, lest IP deliver the second datagram first and confuse the receiver. We discuss this issue more in the next section when we discuss TCP sequence space issues.

IP Fragmentation — Different network media have different limits on the maximum datagram size. This limit is typically referred to as the Maximum Transmission Unit (MTU). When a router is moving a datagram from one media to another, it may discover that the datagram, which was of legal size on the inbound media, is too big for the outbound media. To get around this problem, IP supports fragmentation and reassembly, in which a router can break the datagram up into smaller datagrams to fit on the outbound media. The smaller datagrams are reassembled into the original larger datagram at the destination (not the intermediate hops).

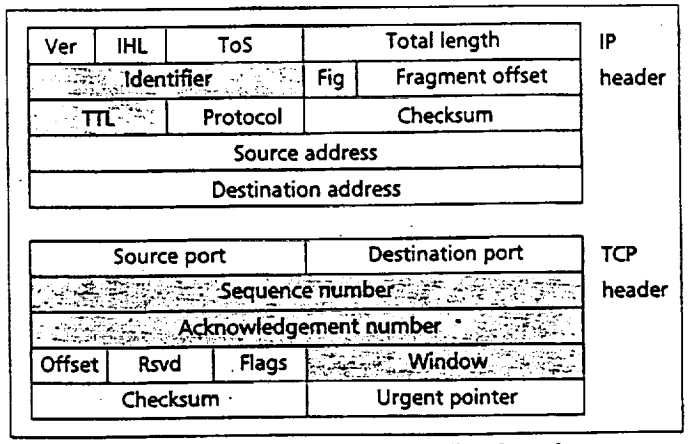
Fragments are identified using a fragment offset field (which indicates the offset of the fragment from the start of the original datagram). Datagrams are uniquely identified by their source, destination, higher layer protocol type, and a 16-bit IP identifier (which must be unique when combined with the source, destination and protocol type).

Observe that there's a clear link between the TTL field and the IP identifier (first identified by [4]). An IP source must ensure that it does not send two datagrams with the same IP identifier to the same destination, using the same protocol within a maximum datagram lifetime, or fragments of two different datagrams may be incorrectly combined. Since the IP identifier is only 16 bits, if the maximum datagram lifetime is two minutes, we are limited to a transmission rate of only 546 datagrams per second. That's clearly not fast enough. The maximum IP datagram size is 64 KB, so 546 datagrams is, at best, a bit less than 300 Mb/s.

The problem of worrying about IP identifier consumption has largely been solved by the development of MTU Discovery a technique for IP sources to discover the MTU of the path to a destination [5]. MTU Discovery is a mechanism that allows hosts to determine the MTU of a path reliably. The existence of MTU discovery allows hosts to set the Don't Fragment (DF) bit in the IP header, to prohibit fragmentation, because the hosts will learn through MTU discovery if their datagrams are too big. Sources that set the DF bit need not worry about the possibility of having two identifiers active at the same time. Systems that do not implement MTU discovery (and thus cannot set the DF bit) need to be careful about this problem.

TCP Throughput Issues

The Transmission Control Protocol (TCP) is the primary transport protocol in the TCP/IP protocol suite. It implements a reliable byte stream over the unreliable datagram service provided by IP. As part of implementing the reliable service, TCP is also responsible for flow and congestion control: ensuring that data is transmitted at a rate consistent with the capacities of both the receiver and the intermediate links in the network path. Since there may be multiple TCP connections active in a link, TCP is also responsible for ensuring that a link's capacity is responsibly shared among the connections using it. As a result, most throughput issues are rooted in TCP.



■ Figure 1. *TCP and IP header fields that affect throughput.*

This section examines the major features of TCP that affect performance. Many of these performance issues have been discovered over the past few years as link transmission speeds have increased and so called high *delay-bandwidth* paths[3] (paths where the product of the path delay and available path bandwidth is big) have become common. To begin to illustrate the challenge, consider that in the 1970s when TCP was being developed, the typical long link was a 56 kb/s circuit across the United States, with a delay-bandwidth product of approximately 0.250 x 56,000 bits or 1.8 KB, while today's Internet contains 2.4 Gb/s circuits crossing the US, which boast a delay-bandwidth product of 75 MB.

Throughput Expectations — Before presenting the performance issues for TCP, it is worth talking briefly about throughput goals.

TCP throughput determines how fast most applications can move data across a network. Application protocols such as HTTP (the World Wide Web protocol), and the File Transfer Protocol (FTP), rely on TCP to carry their data. So TCP performance directly impacts application performance.

While there are no formal TCP performance standards, TCP experts generally expect that, when sending large datagrams (to minimize the overhead of the TCP and IP headers), a TCP connection should be able to fill the available bandwidth of a path and to share the bandwidth with other users. If a link is otherwise idle, a TCP connection is expected to be able to fill it. If a link is shared with three other users, we expect each TCP to get a reasonable share of the bandwidth.

These expectations reflect a mix of practical concerns. When users of TCP acquire faster data lines, they expect their TCP transfers to run faster. And users acquire faster lines for different reasons. Some need faster lines because as their aggregate traffic has increased, they have more applications that need network access. Others have a particular application that requires more bandwidth. The requirement that TCP share a link effectively reflects the needs of aggregation; all users of a faster link should see improvement. The requirement that TCP fill an otherwise idle link reflects the needs of more specialized applications.

TCP Sequence Numbers — TCP keeps track of all data in transit by assigning each byte a unique sequence number. The receiver acknowledges received data by sending an acknowl-

[3] *To avoid confusion, we note that the data networking community, unlike some engineering communities, uses the term bandwidth interchangeably with bitrate.*

edgment which indicates that the receiver has received all data up to a particular byte number.

TCP allocates its sequence numbers from a 32-bit wraparound sequence space. To ensure that a given sequence number uniquely identifies a particular byte, TCP requires that no two bytes with the same sequence number be active in the network at the same time. Recall the early discussion of IP datagram lifetime indicated a datagram was assumed to live for up to two minutes. Thus when TCP sends a byte in an IP datagram, the sequence number of that byte cannot be reused for two minutes. Unfortunately, a 32-bit sequence space spread over two minutes gives a maximum data rate of only 286 Mb/s.

To fix this problem, the Internet End-to-End Research Group devised a set of TCP options and algorithms to extend the sequence space. These changes were adopted by the Internet Engineering Task Force (IETF) and are now part of the TCP standard. The option is a timestamp option [6] which concatenates a timestamp to the 32-bit sequence number. Comparing timestamps using an algorithm called PAWS (Protection Against Wrapped Sequence numbers) makes it possible to distinguish between two identical sequence numbers sent less than two minutes apart.

Depending on the actual granularity of the timestamp (the IETF recommends between 1 second and 1 millisecond), this extension is sufficient for link speeds of between 8 Gb/s and 8 Tb/s (terabits per second).

TCP Transmission Window — The purpose of the transmission window is to allow the receiving TCP to control how much data is being sent to it at any given time. The receiver advertises a window size to the sender. The window measures, in bytes, the amount of unacknowledged data that the sender can have in transit to the receiver. The distinction between the sequence numbers and the window is that sequence numbers are designed to allow the sender to keep track of the data in flight, while the window's purpose is to allow the receiver to control the rate at which it receives data.

Obviously, if a receiver advertises a small window (due, perhaps, to buffer limitations) it is impossible for TCP to achieve high transmission rates. And many implementations do not offer a very large window size (a few kilobytes is typical).

However, there is a more serious problem. The standard TCP window size cannot exceed 64 KB, because the field in the TCP header used to advertise the window is only 16 bits wide. This limits the TCP effective bandwidth to 2^{16} bytes divided by the round-trip time of the path [7]. For long delay links, such as those through satellites with a geosynchronous orbit (GEO), this limit gives a maximum data rate of just under 1 Mb/s.

As part of the changes to add timestamps to the sequence numbers, the End-To-End Research Group and IETF also enhanced TCP to negotiate a window scaling option. The option multiplies the value in the window field by a constant. The effect is that the window can only be adjusted in units of the multiplier. So if the multiplier is 4, an increase of 1 in the advertised window means the receiver is opening the window by 4 bytes.

The window size is limited by the sequence space (the window must be no larger than one half of the sequence space so that it is unambiguously clear that a byte is inside or outside the window). So the maximum multiplier permitted is 2^{14}. This means the maximum window size is 2^{30} and the maximum date rate over a GEO satellite link is approximately 15 Gb/s. Given we have achieved Tb/s data rates in terrestrial fiber, this value is depressingly small, but in the absence of a major change to the TCP header format it is not clear how to fix the problem.

Slow Start — When a TCP connection starts up, the TCP specification requires the connection to be conservative and assume that the available bandwidth to the receiver is small. TCP is supposed to use an algorithm called *slow start* [8], to probe the path to learn how much bandwidth is available.

The slow start algorithm is quite simple and based on data sent per round trip. At the start, the sending TCP sends one TCP segment (datagram) and waits for an acknowledgment. When it gets the acknowledgment, it sends two segments. Many TCPs acknowledge every other segment they receive,[4] so the slow start algorithm effectively sends 50 percent more data every round trip. It continues this process (sending 50 percent more data each round trip) until a segment is lost. This loss is interpreted as indicating congestion and the connection scales back to a more conservative approach (described in the next section) for probing bandwidth for the rest of the connection.

There are two problems with the slow start algorithm on high-speed networks. First, the probing algorithm can take a long time to get up to speed. The time required to get up to speed is $R(1 + \log_{1.5}(DB/l))$, where R is the round-trip time, DB is the delay-bandwidth product and l is the average segment length. If we are trying to fill a pipe with a single TCP connection (and, if the TCP connection is the sole user of the link, filling the link is considered the canonical goal), then DB should be the product of the bandwidth available to the connection and the round-trip time.

An important point is that as the bandwidth goes up or round-trip time increases, or both, this startup time can be quite long. For instance, on a Gb/s GEO satellite link with a 0.5 second round-trip time, it takes 29 round-trip times or 14.5 seconds to finish startup. If the link is otherwise idle, during that period most of the link bandwidth will be unused (wasted).

Even worse is that, in many cases, the entire transfer will complete before the slow start algorithm has finished. The user will never experience the full link bandwidth. All the transfer time will be spent in slow start. This problem is particularly severe for HTTP (the World Wide Web protocol), which is notorious for starting a new TCP connection for every item on a page.[5] This poor protocol design is a (major) reason Web performance on the Internet is perceived as poor: the Web protocols never let TCP get up to full speed.

Currently, the IETF is in the early stages of considering a change to allow TCPs to transmit more than one segment (the current proposal permits between two and four segments) at the beginning of the initial slow start. If there is capacity in the path, this change will reduce the slow start by up to three round-trip times. This change mostly benefits shorter transfers that never get out of slow start.

The second problem is interpreting loss as indicating congestion. TCP has no easy way to distinguish losses due to transmission errors from losses due to congestion, so it makes the conservative assumption that all losses are due to congestion. However, as was shown in an unpublished experiment at MIT, given the loss of a TCP segment early in the slow start process, TCP will then set its initial estimate of the available bandwidth far too low. And since the probing algorithm becomes linear rather than exponential after the initial estimate is set, the time to get to full transmission rate can be very long. On a gigabit GEO link, it could be several hours!

[4] TCP acknowledgments are cumulative, so one acknowledgment can acknowledge multiple segments. Sending one acknowledgment for every two segments reduces the return path bandwidth consumed by the acknowledgments.

[5] A problem now being alleviated by the HTTP 1.1 specification [9].

	1.5 Mb/s			45 Mb/s			155 Mb/s		
	LAN	LEO	GEO	LAN	LEO	GEO	LAN	LEO	GEO
Requires PAWS	No	No	No	No	No	No	Yes	Yes	Yes
Requires large windows	No	No	Yes	No	Yes	Yes	Yes	Yes	Yes
Slow start time	0.01s	1.8s	5.6s	0.2s	3.5s	9.8s	1.9s	4.1s	11.3s
Slow start data (in bytes)	1,760	76,600	197,870	115,900	2,405,000	6,003,000	4,123,814	8,292,000	20,650,000

■ Table 1. *Summary of satellite and TCP interactions.*

Congestion Avoidance — Throughout a TCP connection, TCP runs a congestion avoidance algorithm which is similar to the slow start algorithm and was described in the same paper by Jacobson [8]. Essentially, the sending TCP maintains a congestion window, an estimate of the actual available bandwidth of the path to the receiver. This estimate is set initially by the slow start at the start of the connection. Then the estimate is varied up and down during the life of the connection based on indications of congestion (or the absence thereof). In general, congestion is assumed to be indicated by loss of one or more datagrams.

The basic estimation algorithm is as follows. Every round trip, the sending TCP increases its estimate of the available bandwidth by one maximum-sized segment. Whenever the sender either finds a segment was lost (conservatively assumed to be due to congestion) or receives an indication from the network (e.g., an ICMP Source Quench) that congestion exists, the sender halves its estimate of the available bandwidth. The sender then resumes the one segment per round-trip probing algorithm. (In certain, extreme, loss situations, the sender will do a slow start).

Like the slow start algorithm, the major issue with this algorithm is that over high-delay-bandwidth links, a datagram lost to transmission error will trigger a low estimate of the available bandwidth, and the linear probing algorithm will take a long time to recover.

Another issue is that the rate of improvement under congestion avoidance is a function of the delay-bandwidth product. Basically congestion avoidance allows a sender to increase its window by one segment, for every round-trip time's worth of data sent. In other words, congestion avoidance increases the transmission rate by *1/DB* each round trip [10, 11].

Selective Acknowledgments — Recently the Internet Engineering Task Force has approved an extension to TCP called Selective Acknowledgments (SACKs) [12]. SACKs make it possible for TCP to acknowledge data received out of order. Previously TCP had only been able to acknowledge data received in order.

SACKs have two major benefits. First, they improve the efficiency of TCP retransmissions by reducing the retransmission period. Historically, TCP has used a retransmission algorithm that emulates selective-repeat ARQ using the information provided by in-order acknowledgments. This algorithm works, but takes roughly one round-trip time per lost segment to recover. SACK allows a TCP to retransmit multiple missing segments in a round trip. Second, and more importantly, work by Mathis and Mahdavi [12] has shown that with SACKs a TCP can better evaluate the available path bandwidth in a period of successive losses and avoid doing a slow start.

Inter-Relations — It is important to keep in mind that all the various TCP mechanisms are interrelated, especially when applied to problems of high performance. If the sequence space and window size are not large enough, no improvement to congestion windows will help, since TCP cannot go fast enough anyway. Also, if the receiver chooses a small window size, it takes precedence over the congestion window, and can limit throughput.

More broadly, tinkering with TCP algorithms tends to show odd interrelations. For instance, the individual TCP Vegas performance improvements [13, 14] were shown to work only when applied together applying only some of the changes actually degraded performance. And there are also known TCP syndromes where the congestion window gets misestimated, causing the estimation algorithm to briefly thrash before converging on a congestion window. (The best known is a case where a router has too little buffer space, causing bursts of datagrams to be lost even though there is link capacity to carry all the datagrams).

Satellites and TCP/IP Throughput

For the rest of this article we apply the general discussion of the previous section to the specific problem of achieving high throughput over satellite links. First, we point out the need to implement the extensions to the TCP sequence space and window size. Then we discuss the relationship between slow start and performance over satellite links and some possible solutions.

Currently satellites offer a range of channel bandwidths, from the very small (a compressed phone circuit of a few kb/s) to the very large (the Advanced Communications and Telecommunications Satellite with 622-Mb/s circuits). They also have a range of delays, from relatively small delays of low earth orbit (LEO) satellites to the much larger delays of GEO satellites. Our concern is making TCP/IP work well over those ranges.

General Performance

Many of the problems described in the previous section on TCP/IP performance were ones that became acute only over high-delay-bandwidth paths. One of the first things to note is that all but the slowest satellite links are, by definition, high-delay-bandwidth paths, because the transmission delays to and from the satellite from the Earth's surface are large.

Table 1 illustrates for a range of common bandwidths, when the TCP enhancements of PAWS and large windows are required to fully utilize the bandwidth on a LAN link with 5 ms one-way delay, a LEO link (100 ms one-way) and GEO (250 ms one-way) link, for a range of link speeds. We also indicate how long slow start takes to get to full link speed, assuming 1 KB datagrams (a typical size) are transmitted and how much data is transferred during the slow start phase.

The table highlights some key challenges for satellites (and also for transcontinental terrestrial links, which have delays similar to LEO satellite links). One simply cannot get a TCP/IP implementation to perform well at higher speeds unless it supports large windows, and at speeds past about 100 Mb/s, PAWS. Thus anyone who has not had their TCP/IP software upgraded with PAWS and large windows will not be able to achieve high performance over a satellite link.

Buffer Size in segments	p	Link Rates		
		1.5 Mb/s	45 Mb/s	155 Mb/s
10	4	3×10^6	9×10^7	3.1×10^8
100	13	9.8×10^6	2.9×10^8	1×10^9
1000	44	3.3×10^7	9.9×10^8	3.4×10^9

■ Table 2. *Approximate number of bits sent over GEO link during congestion avoidance.*

Slow Start Revisited

Another point of Table 1 is that the initial slow start period can be quite long and involve large quantities of data. Particularly striking is the column for 155 Mb/s transfers. Between 8 and 21 megabytes of data are sent over a satellite link during slow start at 155 Mb/s. Even at 1.5 Mb/s a GEO link must carry nearly 200 KB before slow start ends. Few data transfers on the Internet are megabytes long. Many are a few kilobytes. All of which says that satellite links will look slow and inefficient for the average data transmission. Interestingly enough, long-distance terrestrial links will also look slow. Their delays are comparable to those of LEO links.

Furthermore, observe that the table helps explain the variation in reported TCP goodput over satellite links. Short data transfers will never achieve full link rate. In many cases, a gigabyte file transfer or larger is probably required to ensure throughput figures are not heavily influenced by slow start.

Obviously some sort of solution to reduce the slow start transient would be desirable. But finding a solution isn't easy.

One obvious solution is to dispense with slow start and just start sending as fast as one can until data is dropped, and then slow down. This approach is known to be disasterous. Indeed, slow start was invented in an environment in which TCP implementations behaved this way and were driving the Internet into congestion collapse. As one example of how this scheme goes wrong, consider a Gb/s capable TCP launching several 100s of megabits of data over a path that turns out to have only 9.6 kb/s of bandwidth. There's a tremendous bandwidth mismatch which will cause datagrams to be discarded or suffer long queuing delays.

As this example illustrates, one of the important problems is that a sending TCP has no idea, when it starts sending, how much bandwidth a particular transmission path has. In the absence of knowledge, a TCP should be conservative. And slow start is conservative — it starts by sending just one datagram in the first round trip.

However, it is clear that somehow we need to be able to give TCP more information about the path if we are to avoid the peril of having TCP chronically spend its time in slow start. One nice aspect of this problem is that it is not specific to satellites. Terrestrial lines need a solution too, and thus if we can find a general solution that works for both satellites and terrestrial lines, everyone will be happy to adopt it.

Improving Slow Start — If the TCP had more information about the path, it could presumably skip at least some of the slow start process possibly by starting the slow start at a somewhat higher rate than one datagram. (The IETF initiative to use a slightly larger beginning transmission size for the initial slow start is a step in this direction). But actually learning the properties of the path is hard. IP keeps no path bandwidth information, so TCP cannot ask the network about path properties. And while there are ways to estimate path bandwidth dynamically, such as packet-pair [12, 13], the estimates can easily be distorted in the presence of cross traffic.

TCP Spoofing — Another idea for getting around slow start is a practice known as "TCP spoofing," described in [14]. The idea calls for a router near the satellite link to send back acknowledgments for the TCP data to give the sender the illusion of a short delay path. The router then suppresses acknowledgments returning from the receiver, and takes responsibility for retransmitting any segments lost downstream of the router.

There are a number of problems with this scheme. First, the router must do a considerable amount of work after it sends an acknowledgment. It must buffer the data segment because the original sender is now free to discard its copy (the segment has been acknowledged) and so if the segment gets lost between the router and the receiver, the router has to take full responsibility for retransmitting it. One side effect of this behavior is that if a queue builds up, it is likely to be a queue of TCP segments that the router is holding for possible retransmission. Unlike IP datagrams, this data cannot be deleted until the router gets the relevant acknowledgments from the receiver.

Second, spoofing requires symmetric paths: the data and acknowledgments must flow along the same path through the router. However, in much of the Internet, asymmetric paths are quite common [15].

Third, spoofing is vulnerable to unexpected failures. If a path changes or the router crashes, data may be lost. Data may even be lost after the sender has finished sending and, based on the router's acknowledgments, reported data successfully transferred.

Fourth, it doesn't work if the data in the IP datagram is encrypted because the router will be unable to read the TCP header.

Cascading TCP — Cascading TCP, also know as split TCP, is a idea where a TCP connection is divided into multiple TCP connections, with a special TCP connection running over the satellite link. The thought behind this idea is that the TCP running over the satellite link can be modified, with knowledge of the satellite's properties, to run faster.

Because each TCP connection is terminated, cascading TCP is not vulnerable to asymmetric paths. And in cases where applications actively participate in TCP connection management (such as Web caching) it works well. But otherwise cascading TCP has the same problems as TCP spoofing.

Error Rates for Satellite Paths

Experience suggests that satellite paths have higher error rates than terrestrial lines. In some cases, the error rates are as high as 1 in 10^{-5}.

Higher error rates matter for two reasons. First, they cause errors in datagrams, which will have to be retransmitted. Second, as noted above, TCP typically interprets loss as a sign of congestion and goes back into a modified version of slow start. Clearly we need to either reduce the error rate to a level acceptable to TCP or find a way to let TCP know that the datagram loss is due to transmission errors, not congestion (and thus TCP should not reduce its transmission rate).

Acceptable Error Rates — What is an acceptable link error rate in a TCP/IP environment? There is no hard and fast answer to this problem. This section presents one way to think about the problem for satellites: looking at TCP's natural frequency of congestion avoidance starts, and seeking an error rate that is substantially less than that frequency.

Suppose we consider the performance of a single established TCP over an otherwise idle link. Once past the initial slow start, the established TCP connection with data to send will alternate between two modes:

- Performing congestion avoidance until a segment is dropped, at which point the TCP falls back to half its window size and resumes congestion avoidance

• Occasionally performing a slow start when loss becomes severe.

During much of the congestion avoidance phase, the TCP will typically be using the path at or near full capacity. Roughly speaking this phase lasts p round-trip times, where p is the largest value such that the following inequality is true:

$$\sum_{j=1}^{p} i \le b$$

where b is the buffering in segments at the bottleneck in the path. (Why this equation? In congestion avoidance the TCP is sending an additional segment every round trip. Suppose we start congestion avoidance at exactly the right window size, namely the delay-bandwidth product. In the first round trip of congestion avoidance the TCP will be sending one segment more than the capacity of the path, so this segment will end up sitting in a queue. In the second round trip, the TCP will send two segments more than the capacity and these two segments will join the first one segment in the queue. And so forth, until the queue is filled and a segment is dropped.) Table 2 shows the number of bits sent during the congestion avoidance phase for a range of GEO link speeds, buffer sizes and values of p.

Clearly we would like to avoid terminating the congestion avoidance phase early, since it causes TCP to underestimate the available bandwidth. Turning this point around, we can say that a link should have an effective error rate sufficiently low that it is very unlikely that the congestion avoidance phase will be prematurely ended by a transmission error. Table 2 suggests this requirement means that satellite error rates on higher-speed links need to be on the order of 1 in 10^{12} or better. That's about the edge of the projected error rates for new satellites. The ACTS satellite routinely sends 10^{13} bits of data without an error. Proposed Ka band systems are aiming for an effective error rate of about 1 in 10^{12}.

Teaching TCP to Ignore Transmission Errors — As an alternative to, or in conjunction with, reducing satellite error rates we might wish to teach TCP to be more intelligent about handling transmission errors. There are basically two approaches: either TCP can explicitly be told that link errors are occurring or TCP can infer that link errors are occurring.

NASA has funded some experiments with explicit error notification as part of a broader study on very long space links done at Mitre [16]. One general challenge in explicit notification is that TCP and IP rarely know that transmission errors have occurred because transmission layers discard the errored datagrams without passing them to TCP and IP.

Having TCP infer which errors are due to transmission errors rather than congestion also presents challenges. One has to find a way for TCP to distinguish congestion from transmission errors reliably, using only information provided by TCP acknowledgments. And the algorithm better never make a mistake, because a failure to respond to congestion loss can exacerbate network congestion. So far as we know, no one has experimented with inferring transmission errors.

Conclusions

Satellite links are today's high-delay-bandwidth paths. Tomorrow high-delay-bandwidth paths will be everywhere. (Consider that some carriers are already installing terrestrial OC-768 [40 Gb/s] network links.) So most of the problems described in this article need to be solved not just for satellites but for high-delay paths in general.

The first step to achieving high performance is making sure the sending and receive TCP implementations contain all the modern features (large windows, PAWS, and SACK) and that the TCP window space is larger than the delay-bandwidth product of the path. Any user worried about high performance should take these steps now.

The next step is to find ways to further improve the performance of TCP over long delay paths and in particular, reduce the impact of slow start. Slow start provides an essential service; the issue is whether there are ways to reduce its start up time, especially when the connection first starts. Because long delay satellite links are only an instance of the larger problem of high-delay bandwidth paths, the authors are less interested in point solutions that only address the performance problems for satellites. We look with hope for solutions that benefit both terrestrial and satellite links.

References

[1] D. E. Comer, *Internetworking with TCP/IP, Vol. I: Principles, Protocols and Architecture, 2nd ed.*, Prentice Hall, 1991.
[2] W. R. Stevens, *TCP/IP Illustrated, Vol. I*, Addison Wesley, 1994.
[3] J. Postel, "Internet Protocol; RFC-791," Internet Requests for Comments, no. 791, Sept. 1981.
[4] C. A. Kent and J. C. Mogul, "Fragmentation Considered Harmful," *Proc. of ACM SIGCOMM '87*, Stowe, VT, 11–13, Aug. 1987, pp. 390–401.
[5] J. Mogul and S. Deering, "Path MTU Discovery; RFC-1191," Internet Requests for Comments, no. 1191, Nov. 1990.
[6] D. Borman, R. Braden, and V. Jacobson, "TCP Extensions for High Performance; RFC-1323," Internet Requests for Comments, no. 1323, May 1992.
[7] A. McKenzie, "Problem with the TCP Big Window Option; RFC-1110," Internet Requests for Comments, no. 1110, Aug. 1989.
[8] V. Jacobson, "Congestion Avoidance and Control," *Proc. ACM SIGCOMM '88*, Stanford, CA, Aug. 1988, pp. 314–329.
[9] H. F. Nielsen *et al.*, "Network Performance Effects of HTTP/1.1, CSS1, and PNG," *Proc. ACM SIGCOMM '97*, Sept. 1997.
[10] S. Floyd and V. Jacobson, "On Traffic Phase Effects in Packet-Switched Gateways," *Internetworking: Research and Experience*, vol. 3, no. 3, Sept. 1992.
[11] S. Floyd, Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic., 21, *Computer Communication Review*, Oct. 1991.
[12] M. Mathis and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control," *Proc. ACM SIGCOMM '96*, Aug. 1996, pp. 281–291.
[13] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Avoidance and Control," *Proc. ACM SIGCOMM '94*, Aug. 1994, pp. 24–35.
[14] Z. Liu *et al.*, "Evaluation of TCP Vegas: Emulation and Experiment," *Proc. ACM SIGCOMM '95*, Aug. 1995, pp. 185–196.
[15] S. Keshav, "A Control-Theoretic Approach to Flow Control," *Proc. ACM SIGCOMM '91*, Zurich, Sept. 1991, pp. 3–16.
[16] J. C. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP," *Proc. ACM SIGCOMM '97*, Aug. 1996, pp. 270–280.
[17] Y. Zhang *et al.*, "Satellite Communications in the Global Internet—Issues, Pitfalls, and Potential," *Proc. INET '97*, 1997.
[18] V. Paxson, "End-to-End Routing Behavior in the Internet," *Proc. ACM SIGCOMM '97*, Aug. 1996, pp. 25–38.
[19] R. C. Durst, G. J. Miller, and E. J. Travis, "TCP Extensions for Space Communications," *Proc. ACM MobiComm '97*, Nov. 1996.

Additional Reading

[1] M. Mathis, J. Mahdavi, and S. Floyd, A. Romanow, and TCP Selective Acknowledgments Options; RFC-2018, Internet Requests for Comments, no. 2018, Oct. 1996.
[2] M. Allman *et al.*, "TCP Performance Over Satellite Links," Proc. Fifth Intl. Conf. on Telecommunications Systems, Nashville, TN, March 1997.
[3] T. V. Lakshman and U. Madhow, "Window-Based Congestion Control in Networks with High Bandwidth-Delay Products," Proc. 3rd ORSA Telecommunications Conference, March 1995.

Biographies

CRAIG PARTRIDGE [SM] (craig@bbn.com) is a Principal Scientist at BBN Technologies where he does research on gigabit and terabit networks. He is the former Editor-in-Chief of *IEEE Network* and *ACM Computer Communication Review*. He is also a consulting associate professor at Stanford University and received his Ph.D. from Harvard University.

TIMOTHY SHEPARD [M] (shep@bbn.com) is a Scientist at BBN Technologies. While a student at MIT, he studied the performance behavior of TCP implementations, which led to the development of a graphical method of TCP packet trace analysis. His interests are in the engineering of large-scale complex systems, particularly those involving microwaves and millions of computers.